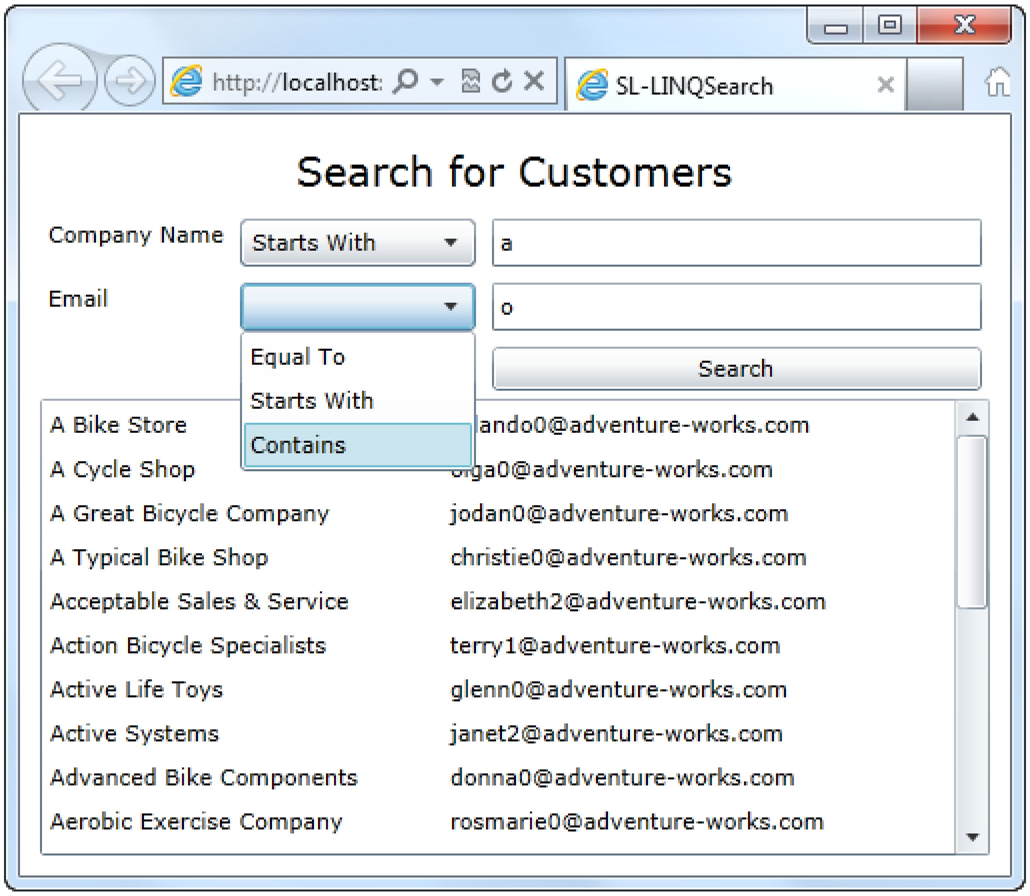
Dynamic Search with LINQ, the Entity Framework and Silverlight

I have been helping a client with a Silverlight application and one of his requirements was to allow his users to be able to query 1 to 5 fields and use different operators for each field. For example, they can choose to search for a Company Name that "Starts With" a certain value and also search for an Email field that "Contains" another value. You can see an example of this search screen in Figure 1.



Search for Customers

Company Name	Starts With	a
Email		o

A Bike Store	lando0@adventure-works.com
A Cycle Shop	ga0@adventure-works.com
A Great Bicycle Company	jodan0@adventure-works.com
A Typical Bike Shop	christie0@adventure-works.com
Acceptable Sales & Service	elizabeth2@adventure-works.com
Action Bicycle Specialists	terry1@adventure-works.com
Active Life Toys	glenn0@adventure-works.com
Active Systems	janet2@adventure-works.com
Advanced Bike Components	donna0@adventure-works.com
Aerobic Exercise Company	rosmarie0@adventure-works.com

Figure 1: A search screen where the user can select an operation and a value for the searching on multiple fields.

To make this search screen work you must pass two values for each search parameter you want to use. You need the operator to use and the value to search for. In this example I pass 4 parameters to a WCF Service. You might modify this method to pass in a collection of objects that contain the different values for each search.

In the code listed below you can see the Click event procedure behind the Search button on the screen. I did not use a View Model for this simple example since the point of this blog has to do with a dynamic LINQ search on the back end, but feel free to add your own View Model class.

```
C#
private CustomerSearchClient _Client = null;

private void btnGetCustomers_Click(object sender,
 RoutedEventArgs e)
{
  _Client = new CustomerSearchClient();

  _Client.GetCustomersCompleted += new
      EventHandler<GetCustomersCompletedEventArgs>
       (_Client_GetCustomersCompleted);

  _Client.GetCustomersAsync(txtCompanyName.Text,
        ((ComboBoxItem)cboCompanyOperator.SelectedItem)
            .Content.ToString(),
        txtEmail.Text,
        ((ComboBoxItem)cboEmailOperator.SelectedItem)
            .Content.ToString());
}

void _Client_GetCustomersCompleted(object sender,
 GetCustomersCompletedEventArgs e)
{
  lstCustomers.DataContext = e.Result;

  _Client.CloseAsync();
}

Visual Basic
Private WithEvents _Client As CustomerSearchClient

Private Sub btnGetCustomers_Click(sender As System.Object, _
  e As System.Windows.RoutedEventArgs) _
   Handles btnGetCustomers.Click
  _Client = New CustomerSearchClient()

  _Client.GetCustomersAsync(txtCompanyName.Text, _
           DirectCast(cboCompanyOperator.SelectedItem, _
              ComboBoxItem).Content.ToString(), _
           txtEmail.Text, _
           DirectCast(cboEmailOperator.SelectedItem, _
              ComboBoxItem).Content.ToString())
End Sub

Private Sub _Client_GetCustomersCompleted(sender As Object, _
 e As _
    CustomerServiceReference.GetCustomersCompletedEventArgs) _
      Handles _Client.GetCustomersCompleted
  lstCustomers.DataContext = e.Result

  _Client.CloseAsync()
End Sub
```

In the code above you create an instance of a WCF Service reference that calls the method named GetCustomers(). This method takes the company name value, the operator for how to search for the company name, the email value and the operator for how to search for the email. These four values are simply taken from the appropriate controls on this Silverlight user control.

Building the Dynamic LINQ Search

To dynamically build a LINQ search from the 4 values passed into the WCF Service you add an ADO.NET Entity Data Model to query against. In this sample I used the **AdventureWorksLT** database and added the **Customer** table to my Entity Data Model. I named this Entity Data Model **AdvWorks**. I then built a WCF Service named **CustomerSearch** and added the GetCustomers() method with the 4 parameters. You will need to add a using/Imports to the System.Data.Objects namespace in order to use the ObjectQuery class.

The ObjectQuery class allows you to use your Entity Framework context classes within a string to express your query. The complete code for the GetCustomers() method is listed further below, but let me give you just a simple little sample of how this works. Take the example below:

```
C#
AdventureWorksLTEntities db =
  new AdventureWorksLTEntities();

ObjectQuery<Customer> query = null;

sql = "SELECT VALUE cust FROM
         AdventureWorksLTEntities.Customers As cust ";
sql += " ORDER BY cust.CompanyName";

query = db.CreateQuery<Customer>(sql);

Visual Basic
Dim db As New AdventureWorksLTEntities

Dim query As ObjectQuery(Of Customer)

sql = "SELECT VALUE cust FROM
         AdventureWorksLTEntities.Customers As cust "
sql &= " ORDER BY cust.CompanyName"

query = db.CreateQuery(Of Customer)(sql)
```

In the above code you use the **AdventureWorksLTEntities** class which is created by the Entity Framework when you add a Data Model to the AdventureWorksLT database. A **Customers** collection object is created within this class to hold a collection of Customer objects. You must use these names within your string object. You also need to use the keyword "VALUE" followed by an alias name, as I used "cust" in the above example. If you will need to reference any specific column names within a WHERE clause or an ORDER BY clause you will reference those column names using this alias.

Once you have created the SQL string you use your AdventureWorksLTEntities object, the variable ***db*** in the code above, and call the CreateQuery() method passing in the SQL string you created. This will build a collection of Customer objects by executing this query against the database. You can view the query that is expressed by turning on the SQL Profiler and tracing any T-SQL calls to the database.

The complete GetCustomers() method builds the SQL statement dynamically by checking if the company name parameter or the email parameter is filled in. If they are filled in then an appropriate WHERE clause is added to the SELECT statement. You can now look at the complete GetCustomers() method below:

```
C#
using System.Collections.Generic;
using System.Data.Objects;
using System.Linq;

public class CustomerSearch : ICustomerSearch
{
  public List<Customer> GetCustomers(string cname,
     string cnameOperator,
     string email,
     string emailOperator)
  {
    AdventureWorksLTEntities db =
       new AdventureWorksLTEntities();
    string join = " WHERE ";
    string sql = null;
    ObjectQuery<Customer> query = null;

    sql = "SELECT VALUE cust FROM
           AdventureWorksLTEntities.Customers As cust ";

    if (string.IsNullOrEmpty(cname) == false)
    {
      sql += join + " cust.CompanyName " +
             BuildWhere(cnameOperator, cname);
      join = " AND ";
    }
    if (string.IsNullOrEmpty(email) == false)
    {
      sql += join + " cust.EmailAddress " +
             BuildWhere(emailOperator, email);
      join = " AND ";
    }

    sql += " ORDER BY cust.CompanyName";

    query = db.CreateQuery<Customer>(sql);

    return query.ToList();
  }

  public string BuildWhere(string operatorValue, string value)
  {
    string where = string.Empty;

    switch (operatorValue.ToLower())
    {
      case "equal to":
        where = " = '" + value + "'";
        break;
      case "starts with":
        where = " LIKE '" + value + "%'";
        break;
      case "contains":
        where = " LIKE '%" + value + "%'";
        break;
    }

    return where;
  }
}

Visual Basic
Imports System.Data.Objects

Public Class CustomerSearch
  Implements ICustomerSearch

  Public Function GetCustomers(cname As String, _
        cnameOperator As String, _
        email As String, _
        emailOperator As String) As List(Of Customer) _
          Implements ICustomerSearch.GetCustomers

    Dim db As New AdventureWorksLTEntities
    Dim join As String = " WHERE "
    Dim sql As String
    Dim query As ObjectQuery(Of Customer)

    sql = "SELECT VALUE cust FROM
           AdventureWorksLTEntities.Customers As cust "

    If String.IsNullOrEmpty(cname) = False Then
      sql &= join & " cust.CompanyName " & _
             BuildWhere(cnameOperator, cname)
      join = " AND "
    End If
    If String.IsNullOrEmpty(email) = False Then
      sql &= join & " cust.EmailAddress " & _
             BuildWhere(emailOperator, email)
      join = " AND "
    End If

    sql &= " ORDER BY cust.CompanyName"

    query = db.CreateQuery(Of Customer)(sql)

    Return query.ToList()
  End Function

  Public Function BuildWhere(operatorValue As String, _
      value As String) As String
    Dim where As String = String.Empty

    Select Case operatorValue.ToLower()
      Case "equal to"
        where = " = '" & value & "'"

      Case "starts with"
        where = " LIKE '" & value & "%'"

      Case "contains"
        where = " LIKE '%" & value & "%'"
    End Select

    Return where
  End Function
End Class
```

To build the WHERE clause you notice that I pass in the operator such as “Equal To”, “Starts With”, or “Contains” to a method called BuildWhere(). This method builds the expression for the WHERE clause. If the value of the operator is “Equal To”, then the equal sign (=) followed by the exact value typed into the text box wrapped into single quotes is returned. If the operator passed in is “Starts With”, then a LIKE operator followed by a single quote, the value typed into the text box, and a percent sign (%) and a closing single quote is returned. It the operator is “Contains”, then a percent sign is wrapped on both sides of the value typed into the text box with a LIKE operator.

All of this will build a SELECT statement that might look like one the following:

```
SELECT VALUE cust
   FROM AdventureWorksLTEntities.Customers As cust
   WHERE  cust.CompanyName  LIKE 'a%'
   ORDER BY cust.CompanyName

or

SELECT VALUE cust
   FROM AdventureWorksLTEntities.Customers As cust
   WHERE  cust.CompanyName  LIKE '%a%'
   AND    cust.EmailAddress LIKE 'o%'
   ORDER BY cust.CompanyName

or

SELECT VALUE cust
   FROM AdventureWorksLTEntities.Customers As cust
   WHERE  cust.EmailAddress  LIKE 'a%'
   ORDER BY cust.CompanyName
```

Summary

While there are other approaches to this problem, I really like this one, because it helps me control the SQL that the Entity Framework submits to the back end database. When using LINQ, sometimes the SQL that the Entity Framework can be pretty convoluted. Using the CreateQuery() method I can sometimes craft the SQL a little closer to what will be eventually submitted to the back end and this can really improve the performance in a lot of cases. I hope you will find this little trick helpful.

NOTE: You can download the sample code for this article by visiting my website at http://www.pdsa.com/downloads. Select “Tips & Tricks”, then select “Dynamic Search with LINQ, the Entity Framework and Silverlight” from the drop down list.